知道创宇

密级 ★ | 项目组公开

智能合约审计报告

安全状态

安全

★ ★ ★ ★ ★

主测人： 知道创宇区块链安全研究团队

版本说明

修订内容	时间	修订者	版本号
编写文档	20210326	知道创宇区块链安全研究团队	V1.0

文档信息

文档名称	文档版	报告编号	保密级别
YouSwap 智能合约审计报告	V1.0		项目组公开

声明

创宇仅就本报告出具前已经发生或存在的事实出具本报告，并就此承担相应责任。对于出具以后发生或存在的事实，创宇无法判断其智能合约安全状况，亦不对此承担责任。本报告所作的安全审计分析及其他内容，仅基于信息提供者截至本报告出具时向创宇提供的文件和资料。创宇假设:已提供资料不存在缺失、被篡改、删减或隐瞒的情形。如已提供资料信息缺失、被篡改、删减、隐瞒或反映的情况与实际情况不符的，创宇对由此而导致的损失和不利影响不承担任何责任。

目录

1. 综述

本次报告有效测试时间是从 2021 年 3 月 24 日开始到 2021 年 3 月 26 日结束，在此期间针对 **YouSwap 智能合约代码**的安全性和规范性进行审计并以此作为报告统计依据。

本次智能合约安全审计的范围，不包含外部合约调用，不包含未来可能出现的新型攻击方式，不包含合约升级或篡改后的代码(随着项目方的发展，智能合约可能会增加新的 pool、新的功能模块，新的外部合约调用等)，不包含前端安全与服务器安全。

此次测试中，知道创宇工程师对智能合约的常见漏洞（见第四章节）进行了全面的分析，同时对业务逻辑层面进行审计，未发现存在相关安全风险以及逻辑设错误，故对此合约综合评定为**通过**。

本次智能合约安全审计结果：通过

由于本次测试过程在非生产环境下进行，所有代码均为最新备份，测试过程均与相关接口人进行沟通，并在操作风险可控的情况下进行相关测试操作，以规避测试过程中的生产运营风险、代码安全风险。

本次审计的报告信息：

报告编号：

报告查询地址链接:

本次审计的目标信息：

条目	描述	
项目名称	Youswap	
合约地址	YouSwapFactory	https://ropsten.etherscan.io/address/0x2794bF982476C95755604bad8Bf7280f53f71786#code
	Repurchase	https://ropsten.etherscan.io/address/0xcc9BbCaB68c22CfCb5cCf1C461138aD4C79f0cD3#cod

		e
	YouSwapRouter	https://ropsten.etherscan.io/address/0x121E6174575F776c9A58a92F475b8c1a77eF63DB#code
代码类型	以太坊智能合约代码	
代码语言	Solidity	

合约文件及哈希：

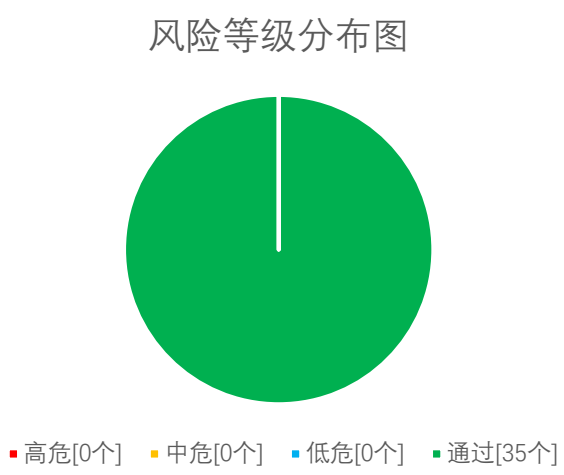
合约文件	MD5
Repurchase.sol	F0E13EE37ED279D27066E2B0693B06A9
IERC20.sol	C0B512FC8612A4A480AEE4CB1FDD89DD
IYouSwapCallee.sol	6D323A9D289D0C7DD213D7D77807A965
IYouSwapERC20.sol	C7AE5CFE6C9E6A90D467A15BB9E9ABC4
IYouSwapFactory.sol	CD95507B6C2EEF2433C93AB7C2378A4F
IYouSwapPair.sol	614E14D8F21E2CF9FB94BCE92526F206
MathV1.sol	683C9332744A5B7400B22F5606FE1D0A
UQ112x112.sol	A9E748B2299564A1FAA22BE501135F2D
YouSwapERC20.sol	A6BA582C125C11FD5C5A53A40319D96B
YouSwapFactory.sol	44C25F80F444CA8B5A192FD9F156DA13
YouSwapPair.sol	9C4C56C05C09AED821EBC9B8AB59CC83
YouSwapRouter.sol	8A63C64773DF9F784FBF4D6268858F24

2. 代码漏洞分析

2.1 漏洞等级分布

本次漏洞风险按等级统计：

安全风险等级个数统计表			
高危	中危	低危	通过
0	0	0	35



2.2 审计结果汇总说明

审计结果			
审计项目	审计内容	状态	描述
业务安全	Caller 增删改查逻辑设计	通过	经检测，不存在该安全问题。
	紧急提款逻辑设计	通过	经检测，不存在该安全问题。
	回购销毁逻辑设计	通过	经检测，不存在该安全问题。
	创建交易对逻辑设计	通过	经检测，不存在该安全问题。
	交易费相关逻辑设计	通过	经检测，不存在该安全问题。
	Mint 函数逻辑设计	通过	经检测，不存在该安全问题。
	Burn 函数逻辑设计	通过	经检测，不存在该安全问题。
	Swap 资产交易逻辑设计	通过	经检测，不存在该安全问题。
编码安全	编译器版本安全	通过	经检测，不存在该安全问题。
	冗余代码	通过	经检测，不存在该安全问题。
	安全算数库的使用	通过	经检测，不存在该安全问题。
	不推荐的编码方式	通过	经检测，不存在该安全问题。
	require/assert 的合理使用	通过	经检测，不存在该安全问题。
	fallback 函数安全	通过	经检测，不存在该安全问题。
	tx.orgin 身份验证	通过	经检测，不存在该安全问题。
	owner 权限控制	通过	经检测，不存在该安全问题。
	gas 消耗检测	通过	经检测，不存在该安全问题。
	call 注入攻击	通过	经检测，不存在该安全问题。
	低级函数安全	通过	经检测，不存在该安全问题。
	增发代币漏洞	通过	经检测，不存在该安全问题。
	访问控制缺陷检测	通过	经检测，不存在该安全问题。
	数值溢出检测	通过	经检测，不存在该安全问题。
	算数精度误差	通过	经检测，不存在该安全问题。

	错误使用随机数检测	通过	经检测，不存在该安全问题。
	不安全的接口使用	通过	经检测，不存在该安全问题。
	变量覆盖	通过	经检测，不存在该安全问题。
	未初始化的存储指针	通过	经检测，不存在该安全问题。
	返回值调用验证	通过	经检测，不存在该安全问题。
	交易顺序依赖	通过	经检测，不存在该安全问题。
	时间戳依赖攻击	通过	经检测，不存在该安全问题。
	拒绝服务攻击	通过	经检测，不存在该安全问题。
	假充值漏洞	通过	经检测，不存在该安全问题。
	重入攻击检测	通过	经检测，不存在该安全问题。
	重放攻击检测	通过	经检测，不存在该安全问题。
	重排攻击检测	通过	经检测，不存在该安全问题。

3. 业务安全性检测

3.1 Caller 增删改查逻辑设计【通过】

对 caller 的增删改查逻辑设计进行审计，检查是否有对参数进行校验、函数调用者权限校验、以及相关逻辑设计是否合理。

审计结果：相关逻辑设计合理无误。

```
function addCaller(address _newCaller) public onlyOwner returns (bool) {
    require(_newCaller != address(0), "NewCaller is the zero address");
    return EnumerableSet.add(_caller, _newCaller);
}

function delCaller(address _delCaller) public onlyOwner returns (bool) {
    require(_delCaller != address(0), "DelCaller is the zero address");
    return EnumerableSet.remove(_caller, _delCaller);
}

function getCallerLength() public view returns (uint256) {
    return EnumerableSet.length(_caller);
}

function isCaller(address _call) public view returns (bool) {
    return EnumerableSet.contains(_caller, _call);
}

function getCaller(uint256 _index) public view returns (address){
    require(_index <= getCallerLength() - 1, "index out of bounds");
    return EnumerableSet.at(_caller, _index);
}
```

安全建议：无。

3.2 紧急提款逻辑设计【通过】

对紧急提款逻辑设计进行审计，检查是否有对参数进行校验、函数调用者权限校验、以及相关逻辑设计是否合理。

审计结果：相关逻辑设计合理无误。

```
    function setEmergencyAddress(address _newAddress) public onlyOwner {
        require(_newAddress != address(0), "Is zero address");
        emergencyAddress = _newAddress;
    }
    function emergencyWithdraw(address _token) public onlyOwner {
        require(IERC20(_token).balanceOf(address(this)) > 0, "Insufficient contract balance");
        IERC20(_token).transfer(emergencyAddress,
IERC20(_token).balanceOf(address(this)));
    }
```

安全建议：无。

3.3 回购销毁逻辑设计【通过】

对回购销毁逻辑设计进行审计，检查是否有对参数进行校验、函数调用者权限校验、以及相关逻辑设计是否合理。

审计结果：相关逻辑设计合理无误。

```
function swap() external onlyCaller returns (uint256 amountOut){
        require(IERC20(USDT).balanceOf(address(this)) >= amountIn, "Insufficient contract
balance");
        (uint256 reserve0, uint256 reserve1,) = IYouSwapPair(YOU_USDT).getReserves();
        uint256 amountInWithFee = amountIn.mul(997);
        amountOut = amountIn.mul(997).mul(reserve0) /
reserve1.mul(1000).add(amountInWithFee);
        _safeTransfer(USDT, YOU_USDT, amountIn);
```

```
        IYouSwapPair(YOU_USDT).swap(amountOut, 0, destroyAddress, new bytes(0));
    }
```

安全建议：无。

3.4 创建交易对逻辑设计【通过】

对创建交易对逻辑设计进行审计，检查是否有对参数进行校验、函数调用者权限校验、以及相关逻辑设计是否合理。

审计结果：相关逻辑设计合理无误。

```
    function createPair(address tokenA, address tokenB) external returns (address pair) {
        require(tokenA != tokenB, 'YouSwap: IDENTICAL_ADDRESSES');
        (address token0, address token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
        require(token0 != address(0), 'YouSwap: ZERO_ADDRESS');
        require(getPair[token0][token1] == address(0), 'YouSwap: PAIR_EXISTS'); // single check is sufficient
        bytes memory bytecode = type(YouSwapPair).creationCode;
        bytes32 salt = keccak256(abi.encodePacked(token0, token1));
        assembly {
            pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
        }
        IYouSwapPair(pair).initialize(token0, token1);
        getPair[token0][token1] = pair;
        getPair[token1][token0] = pair; // populate mapping in the reverse direction
        allPairs.push(pair);
        emit PairCreated(token0, token1, pair, allPairs.length);
    }
```

安全建议：无。

3.5 交易费相关逻辑设计【通过】

对交易费相关逻辑设计进行审计，检查是否有对参数进行校验、函数调用者权限校验、以及相关逻辑设计是否合理。

审计结果：相关逻辑设计合理无误。

```
function setFeeTo(address _feeTo) external {
    require(msg.sender == feeToSetter, 'YouSwap: FORBIDDEN');
    feeTo = _feeTo;
}

function setFeeToSetter(address _feeToSetter) external {
    require(msg.sender == feeToSetter, 'YouSwap: FORBIDDEN');
    feeToSetter = _feeToSetter;
}

function setFeeToRate(uint256 _rate) external {
    require(msg.sender == feeToSetter, 'YouSwap: FORBIDDEN');
    require(_rate > 0, "YouSwap: FEE_TO_RATE_OVERFLOW");
    feeToRate = _rate.sub(1);
}
```

安全建议：无。

3.6 mint 函数逻辑设计【通过】

对 mint 增加流动性代币函数逻辑设计进行审计，检查是否有对参数进行校验、函数调用者权限校验、以及相关逻辑设计是否合理。

审计结果：相关逻辑设计合理无误。

```
// this low-level function should be called from a contract which performs important safety checks
```

```
    function mint(address to) external lock returns (uint liquidity) {
        (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
        uint balance0 = IERC20(token0).balanceOf(address(this));
        uint balance1 = IERC20(token1).balanceOf(address(this));
        uint amount0 = balance0.sub(_reserve0);
        uint amount1 = balance1.sub(_reserve1);

        bool feeOn = _mintFee(_reserve0, _reserve1);
        uint _totalSupply = totalSupply; // gas savings, must be defined here since totalSupply can update in _mintFee
        if (_totalSupply == 0) {
            liquidity = Math.sqrt(amount0.mul(amount1)).sub(MINIMUM_LIQUIDITY);
           _mint(address(0), MINIMUM_LIQUIDITY); // permanently lock the first MINIMUM_LIQUIDITY tokens
        } else {
            liquidity = Math.min(amount0.mul(_totalSupply) / _reserve0, amount1.mul(_totalSupply) / _reserve1);
        }
        require(liquidity > 0, 'YouSwap: INSUFFICIENT_LIQUIDITY_MINTED');
        _mint(to, liquidity);

        _update(balance0, balance1, _reserve0, _reserve1);
        if (feeOn) kLast = uint(reserve0).mul(reserve1); // reserve0 and reserve1 are up-to-date
        emit Mint(msg.sender, amount0, amount1);
    }
```

安全建议：无。

3.7 Burn 函数逻辑设计【通过】

对 burn 流动性销毁函数逻辑设计进行审计，检查是否有对参数进行校验、函数调用者权限校验、以及相关逻辑设计是否合理。

审计结果：相关逻辑设计合理无误。

```
    // this low-level function should be called from a contract which performs important safety checks
    function burn(address to) external lock returns (uint amount0, uint amount1) {
        (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
        address _token0 = token0;                                // gas savings
        address _token1 = token1;                                // gas savings
        uint balance0 = IERC20(_token0).balanceOf(address(this));
        uint balance1 = IERC20(_token1).balanceOf(address(this));
        uint liquidity = balanceOf[address(this)];

        bool feeOn = _mintFee(_reserve0, _reserve1);
        uint _totalSupply = totalSupply; // gas savings, must be defined here since totalSupply can update in _mintFee
        amount0 = liquidity.mul(balance0) / _totalSupply; // using balances ensures pro-rata distribution
        amount1 = liquidity.mul(balance1) / _totalSupply; // using balances ensures pro-rata distribution
        require(amount0 > 0 && amount1 > 0, 'YouSwap: INSUFFICIENT_LIQUIDITY_BURNED');
        _burn(address(this), liquidity);
        _safeTransfer(_token0, to, amount0);
        _safeTransfer(_token1, to, amount1);
        balance0 = IERC20(_token0).balanceOf(address(this));
        balance1 = IERC20(_token1).balanceOf(address(this));

        _update(balance0, balance1, _reserve0, _reserve1);
        if (feeOn) kLast = uint(reserve0).mul(reserve1); // reserve0 and reserve1 are up-to-date
        emit Burn(msg.sender, amount0, amount1, to);
    }
```

安全建议：无。

3.8 swap 资产交易逻辑设计【通过】

对 swap 资产交易函数逻辑设计进行审计，检查是否有对参数进行校验、函数调用者权限校验、以及相关逻辑设计是否合理。

审计结果：相关逻辑设计合理无误。

```
// this low-level function should be called from a contract which performs important safety checks
function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external lock {
require(amount0Out > 0 || amount1Out > 0, 'YouSwap: INSUFFICIENT_OUTPUT_AMOUNT');
(uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
require(amount0Out < _reserve0 && amount1Out < _reserve1, 'YouSwap: INSUFFICIENT_LIQUIDITY');

uint balance0;
uint balance1;
{ // scope for _token{0,1}, avoids stack too deep errors
address _token0 = token0;
address _token1 = token1;
require(to != _token0 && to != _token1, 'YouSwap: INVALID_TO');
if (amount0Out > 0) _safeTransfer(_token0, to, amount0Out); // optimistically transfer tokens
if (amount1Out > 0) _safeTransfer(_token1, to, amount1Out); // optimistically transfer tokens
if (data.length > 0) IYouSwapCallee(to).YouSwapV2Call(msg.sender, amount0Out, amount1Out, data);
balance0 = IERC20(_token0).balanceOf(address(this));
balance1 = IERC20(_token1).balanceOf(address(this));
}
uint amount0In = balance0 > _reserve0 - amount0Out ? balance0 - (_reserve0 -
```

```
amount0Out) : 0;
        uint amount1In = balance1 > _reserve1 - amount1Out ? balance1 - (_reserve1 -
amount1Out) : 0;
        require(amount0In > 0 || amount1In > 0, 'YouSwap:
INSUFFICIENT_INPUT_AMOUNT');
        { // scope for reserve{0,1}Adjusted, avoids stack too deep errors
        uint balance0Adjusted = balance0.mul(1000).sub(amount0In.mul(3));
        uint balance1Adjusted = balance1.mul(1000).sub(amount1In.mul(3));
        require(balance0Adjusted.mul(balance1Adjusted) >=
uint(_reserve0).mul(_reserve1).mul(1000**2), 'YouSwap: K');
        }

        _update(balance0, balance1, _reserve0, _reserve1);
        emit Swap(msg.sender, amount0In, amount1In, amount0Out, amount1Out, to);
    }
```

安全建议：无。

4. 代码基本漏洞检测

4.1. 编译器版本安全【通过】

检查合约代码实现中是否使用了安全的编译器版本

检测结果：经检测，智能合约代码中制定了编译器版本 0.5.15 以上，不存在该安全问题。

安全建议：无。

4.2. 冗余代码【通过】

检查合约代码实现中是否包含冗余代码

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.3. 安全算数库的使用【通过】

检查合约代码实现中是否使用了 SafeMath 安全算数库

检测结果：经检测，智能合约代码中已使用 SafeMath 安全算数库，不存在该安全问题。

安全建议：无。

4.4. 不推荐的编码方式【通过】

检查合约代码实现中是否有官方不推荐或弃用的编码方式

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.5. require/assert 的合理使用【通过】

检查合约代码实现中 require 和 assert 语句使用的合理性

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.6. fallback 函数安全【通过】

检查合约代码实现中是否正确使用 fallback 函数

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.7. tx.origin 身份验证【通过】

tx.origin 是 Solidity 的一个全局变量，它遍历整个调用栈并返回最初发送调用（或事务）的帐户的地址。在智能合约中使用此变量进行身份验证会使合约容易受到类似网络钓鱼的攻击。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.8. owner 权限控制【通过】

检查合约代码实现中的 owner 是否具有过高的权限。例如，任意修改其他

账户余额等。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.9. gas 消耗检测【通过】

检查 gas 的消耗是否超过区块最大限制

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.10. call 注入攻击【通过】

call 函数调用时，应该做严格的权限控制，或直接写死 call 调用的函数。

检测结果：经检测，智能合约未使用 call 函数，不存在此漏洞。

安全建议：无。

4.11. 低级函数安全【通过】

检查合约代码实现中低级函数（call/delegatecall）的使用是否存在安全漏洞

call 函数的执行上下文是在被调用的合约中；而 delegatecall 函数的执行上下文是在当前调用该函数的合约中

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.12. 增发代币漏洞【通过】

检查在初始化代币总量后，代币合约中是否存在可能使代币总量增加的函数。

检测结果：经检测，智能合约代码中存在增发代币的功能，但由于流动性挖矿需要增发代币，故通过。

安全建议：无。

4.13. 访问控制缺陷检测【通过】

合约中不同函数应设置合理的权限

检查合约中各函数是否正确使用了 public、private 等关键词进行可见性修饰，检查合约是否正确定义并使用了 modifier 对关键函数进行访问限制，避免越权导致的问题。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.14. 数值溢出检测【通过】

智能合约中的算数问题是指整数溢出和整数下溢。

Solidity 最多能处理 256 位的数字（2^256-1），最大数字增加 1 会溢出得到 0。同样，当数字为无符号类型时，0 减去 1 会下溢得到最大数字值。

整数溢出和下溢不是一种新类型的漏洞，但它们在智能合约中尤其危险。溢出情况会导致不正确的结果，特别是如果可能性未被预期，可能会影响程序的可靠性和安全性。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.15. 算术精度误差【通过】

Solidity 作为一门编程语言具备和普通编程语言相似的数据结构设计，比如：变量、常量、函数、数组、函数、结构体等等，Solidity 和普通编程语言也有一个较大的区别——Solidity 没有浮点型，且 Solidity 所有的数值运算结果都只会是整数，不会出现小数的情况，同时也不允许定义小数类型数据。合约中的数值运算必不可少，而数值运算的设计有可能造成相对误差，例如同级运算：5/2*10=20，而 5*10/2=25，从而产生误差，在数据更大时产生的误差也会更大，更明显。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.16. 错误使用随机数【通过】

智能合约中可能需要使用随机数，虽然 Solidity 提供的函数和变量可以访问明显难以预测的值，如 block.number 和 block.timestamp，但是它们通常或者比看起来更公开，或者受到矿工的影响，即这些随机数在一定程度上是可预测的，所以恶意用户通常可以复制它并依靠其不可预知性来攻击该功能。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.17. 不安全的接口使用【通过】

检查合约代码实现中是否使用了不安全的接口

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.18. 变量覆盖【通过】

检查合约代码实现中是否存在变量覆盖导致的安全问题

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.19. 未初始化的储存指针【通过】

在 solidity 中允许一个特殊的数据结构为 struct 结构体，而函数内的局部变量默认使用 storage 或 memory 储存。

而存在 storage(存储器)和 memory(内存)是两个不同的概念，solidity 允许指针指向一个未初始化的引用，而未初始化的局部 stroage 会导致变量指向其他储存变量，导致变量覆盖，甚至其他更严重的后果，在开发中应该避免在函数中初始化 struct 变量。

检测结果：经检测，智能合约代码不使用结构体，不存在该问题。

安全建议：无。

4.20. 返回值调用验证【通过】

此问题多出现在和转币相关的智能合约中，故又称作静默失败发送或未经检查发送。

在 Solidity 中存在 transfer()、send()、call.value()等转币方法，都可以用于向

某一地址发送 Ether，其区别在于： transfer 发送失败时会 throw，并且进行状态回滚；只会传递 2300gas 供调用，防止重入攻击；send 发送失败时会返回 false；只会传递 2300gas 供调用，防止重入攻击；call.value 发送失败时会返回 false；传递所有可用 gas 进行调用（可通过传入 gas_value 参数进行限制），不能有效防止重入攻击。

如果在代码中没有检查以上 send 和 call.value 转币函数的返回值，合约会继续执行后面的代码，可能由于 Ether 发送失败而导致意外的结果。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.21. 交易顺序依赖【通过】

由于矿工总是通过代表外部拥有地址（EOA）的代码获取 gas 费用，因此用户可以指定更高的费用以便更快地开展交易。由于以太坊区块链是公开的，每个人都可以看到其他人未决交易的内容。这意味着，如果某个用户提交了一个有价值的解决方案，恶意用户可以窃取该解决方案并以较高的费用复制其交易，以抢占原始解决方案。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.22. 时间戳依赖攻击【通过】

数据块的时间戳通常来说都是使用矿工的本地时间，而这个时间大约能有 900 秒的范围波动，当其他节点接受一个新区块时，只需要验证时间戳是否晚于

之前的区块并且与本地时间误差在 900 秒以内。一个矿工可以通过设置区块的时间戳来尽可能满足有利于他的条件来从中获利。

检查合约代码实现中是否存在有依赖于时间戳的关键功能

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.23. 拒绝服务攻击【通过】

在以太坊的世界中，拒绝服务是致命的，遭受该类型攻击的智能合约可能永远无法恢复正常工作状态。导致智能合约拒绝服务的原因可能有很多种，包括在作为交易接收方时的恶意行为，人为增加计算功能所需 gas 导致 gas 耗尽，滥用访问控制访问智能合约的 private 组件，利用混淆和疏忽等等。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.24. 假充值漏洞【通过】

在代币合约的 transfer 函数对转账发起人(msg.sender)的余额检查用的是 if 判断方式，当 balances[msg.sender] < value 时进入 else 逻辑部分并 return false，最终没有抛出异常，我们认为仅 if/else 这种温和的判断方式在 transfer 这类敏感函数场景中是一种不严谨的编码方式。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.25. 重入攻击检测【通过】

Solidity 中的 call.value()函数在被用来发送 Ether 的时候会消耗它接收到的所有 gas，当调用 call.value()函数发送 Ether 的操作发生在实际减少发送者账户的余额之前时，就会存在重入攻击的风险。

检测结果：经检测，智能合约代码中不存在该安全问题。

安全建议：无。

4.26. 重放攻击检测【通过】

合约中如果涉及委托管理的需求，应注意验证的不可复用性，避免重放攻击

在资产管理体系中，常有委托管理的情况，委托人将资产给受托人管理，委托人支付一定的费用给受托人。这个业务场景在智能合约中也比较普遍。。

检测结果：经检测，智能合约未使用 call 函数，不存在此漏洞。

安全建议：无。

4.27. 重排攻击检测【通过】

重排攻击是指矿工或其他方试图通过将自己的信息插入列表(list)或映射(mapping)中来与智能合约参与者进行“竞争”，从而使攻击者有机会将自己的信息存储到合约中。

检测结果:经检测，智能合约代码中不存在相关漏洞。

安全建议:无。

5. 附录 A：合约代码

本次测试代码来源：

Repurchase.sol

```
/**
 *Submitted for verification at Etherscan.io on 2021-03-17
*/
// File: localhost/contracts/interfaces/IYouSwapPair.sol
pragma solidity >=0.5.0;

interface IYouSwapPair {
    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);

    function name() external pure returns (string memory);
    function symbol() external pure returns (string memory);
    function decimals() external pure returns (uint8);
    function totalSupply() external view returns (uint);
    function balanceOf(address owner) external view returns (uint);
    function allowance(address owner, address spender) external view returns (uint);

    function approve(address spender, uint value) external returns (bool);
    function transfer(address to, uint value) external returns (bool);
    function transferFrom(address from, address to, uint value) external returns (bool);

    function DOMAIN_SEPARATOR() external view returns (bytes32);
    function PERMIT_TYPEHASH() external pure returns (bytes32);
    function nonces(address owner) external view returns (uint);

    function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, bytes32
s) external;
```

```
    event Mint(address indexed sender, uint amount0, uint amount1);
    event Burn(address indexed sender, uint amount0, uint amount1, address indexed to);
    event Swap(
        address indexed sender,
        uint amount0In,
        uint amount1In,
        uint amount0Out,
        uint amount1Out,
        address indexed to
    );
    event Sync(uint112 reserve0, uint112 reserve1);

    function MINIMUM_LIQUIDITY() external pure returns (uint);
    function factory() external view returns (address);
    function token0() external view returns (address);
    function token1() external view returns (address);
    function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32 blockTimestampLast);
    function price0CumulativeLast() external view returns (uint);
    function price1CumulativeLast() external view returns (uint);
    function kLast() external view returns (uint);

    function mint(address to) external returns (uint liquidity);
    function burn(address to) external returns (uint amount0, uint amount1);
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;
    function skim(address to) external;
    function sync() external;

    function initialize(address, address) external;
}

// File: localhost/contracts/libraries/EnumerableSet.sol
```

```
pragma solidity =0.5.16;


/**
 * @dev Library for managing
 * https://en.wikipedia.org/wiki/Set_(abstract_data_type)[sets] of primitive
 * types.
 *
 * Sets have the following properties:
 *
 * - Elements are added, removed, and checked for existence in constant time
 * (O(1)).
 * - Elements are enumerated in O(n). No guarantees are made on the ordering.
 *
 * ```
 * contract Example {
 *         // Add the library methods
 *         using EnumerableSet for EnumerableSet.AddressSet;
 *
 *         // Declare a set state variable
 *         EnumerableSet.AddressSet private mySet;
 * }
 * ```
 *
 * As of v3.3.0, sets of type `bytes32` (`Bytes32Set`), `address` (`AddressSet`)
 * and `uint256` (`UintSet`) are supported.
 */
library EnumerableSet {
        // To implement this library for multiple types with as little code
        // repetition as possible, we write it in terms of a generic Set type with
        // bytes32 values.
        // The Set implementation uses private functions, and user-facing
        // implementations (such as AddressSet) are just wrappers around the
        // underlying Set.
```

```
// This means that we can only create new EnumerableSets for types that fit
// in bytes32.

struct Set {
    // Storage of set values
    bytes32[] _values;

    // Position of the value in the `values` array, plus 1 because index 0
    // means a value is not in the set.
    mapping (bytes32 => uint256) _indexes;
}

/**
 * @dev Add a value to a set. O(1).
 *
 * Returns true if the value was added to the set, that is if it was not
 * already present.
 */
function _add(Set storage set, bytes32 value) private returns (bool) {
    if (!_contains(set, value)) {
        set._values.push(value);
        // The value is stored at length-1, but we add 1 to all indexes
        // and use 0 as a sentinel value
        set._indexes[value] = set._values.length;
        return true;
    } else {
        return false;
    }
}

/**
 * @dev Removes a value from a set. O(1).
 *
```

```
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function _remove(Set storage set, bytes32 value) private returns (bool) {
        // We read and store the value's index to prevent multiple reads from the same storage slot
        uint256 valueIndex = set._indexes[value];

        if (valueIndex != 0) { // Equivalent to contains(set, value)
            // To delete an element from the _values array in O(1), we swap the element to delete with the last one in
            // the array, and then remove the last element (sometimes called as 'swap and pop').
            // This modifies the order of the array, as noted in {at}.

            uint256 toDeleteIndex = valueIndex - 1;
            uint256 lastIndex = set._values.length - 1;

            // When the value to delete is the last one, the swap operation is unnecessary. However, since this occurs
            // so rarely, we still do the swap anyway to avoid the gas cost of adding an 'if' statement.

            bytes32 lastvalue = set._values[lastIndex];

            // Move the last value to the index where the value to delete is
            set._values[toDeleteIndex] = lastvalue;
            // Update the index for the moved value
            set._indexes[lastvalue] = toDeleteIndex + 1; // All indexes are 1-based

            // Delete the slot where the moved value was stored
            set._values.pop();

            // Delete the index for the deleted slot
            delete set._indexes[value];
```

```
            return true;
        } else {
            return false;
        }
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function _contains(Set storage set, bytes32 value) private view returns (bool) {
        return set._indexes[value] != 0;
    }

    /**
     * @dev Returns the number of values on the set. O(1).
     */
    function _length(Set storage set) private view returns (uint256) {
        return set._values.length;
    }

   /**
    * @dev Returns the value stored at position `index` in the set. O(1).
    *
    * Note that there are no guarantees on the ordering of values inside the
    * array, and it may change when more values are added or removed.
    *
    * Requirements:
    *
    * - `index` must be strictly less than {length}.
    */
    function _at(Set storage set, uint256 index) private view returns (bytes32) {
        require(set._values.length > index, "EnumerableSet: index out of bounds");
        return set._values[index];
```

```
}

// Bytes32Set

struct Bytes32Set {
    Set _inner;
}

/**
 * @dev Add a value to a set. O(1).
 *
 * Returns true if the value was added to the set, that is if it was not
 * already present.
 */
function add(Bytes32Set storage set, bytes32 value) internal returns (bool) {
    return _add(set._inner, value);
}

/**
 * @dev Removes a value from a set. O(1).
 *
 * Returns true if the value was removed from the set, that is if it was
 * present.
 */
function remove(Bytes32Set storage set, bytes32 value) internal returns (bool) {
    return _remove(set._inner, value);
}

/**
 * @dev Returns true if the value is in the set. O(1).
 */
function contains(Bytes32Set storage set, bytes32 value) internal view returns (bool) {
    return _contains(set._inner, value);
```

```
    }

    /**
     * @dev Returns the number of values in the set. O(1).
     */
    function length(Bytes32Set storage set) internal view returns (uint256) {
        return _length(set._inner);
    }

/**
 * @dev Returns the value stored at position `index` in the set. O(1).
 *
 * Note that there are no guarantees on the ordering of values inside the
 * array, and it may change when more values are added or removed.
 *
 * Requirements:
 *
 * - `index` must be strictly less than {length}.
 */
function at(Bytes32Set storage set, uint256 index) internal view returns (bytes32) {
    return _at(set._inner, index);
}

// AddressSet

struct AddressSet {
    Set _inner;
}

/**
 * @dev Add a value to a set. O(1).
 *
 * Returns true if the value was added to the set, that is if it was not
```

```
     * already present.
     */
    function add(AddressSet storage set, address value) internal returns (bool) {
        return _add(set._inner, bytes32(uint256(value)));
    }

    /**
     * @dev Removes a value from a set. O(1).
     *
     * Returns true if the value was removed from the set, that is if it was
     * present.
     */
    function remove(AddressSet storage set, address value) internal returns (bool) {
        return _remove(set._inner, bytes32(uint256(value)));
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function contains(AddressSet storage set, address value) internal view returns (bool) {
        return _contains(set._inner, bytes32(uint256(value)));
    }

    /**
     * @dev Returns the number of values in the set. O(1).
     */
    function length(AddressSet storage set) internal view returns (uint256) {
        return _length(set._inner);
    }

    /**
     * @dev Returns the value stored at position `index` in the set. O(1).
     *
```

```
* Note that there are no guarantees on the ordering of values inside the
* array, and it may change when more values are added or removed.
*
* Requirements:
*
* - `index` must be strictly less than {length}.
*/
function at(AddressSet storage set, uint256 index) internal view returns (address) {
    return address(uint256(_at(set._inner, index)));
}


// UintSet

struct UintSet {
    Set _inner;
}

/**
 * @dev Add a value to a set. O(1).
 *
 * Returns true if the value was added to the set, that is if it was not
 * already present.
 */
function add(UintSet storage set, uint256 value) internal returns (bool) {
    return _add(set._inner, bytes32(value));
}

/**
 * @dev Removes a value from a set. O(1).
 *
 * Returns true if the value was removed from the set, that is if it was
 * present.
```

```
     */
    function remove(UintSet storage set, uint256 value) internal returns (bool) {
        return _remove(set._inner, bytes32(value));
    }

    /**
     * @dev Returns true if the value is in the set. O(1).
     */
    function contains(UintSet storage set, uint256 value) internal view returns (bool) {
        return _contains(set._inner, bytes32(value));
    }

    /**
     * @dev Returns the number of values on the set. O(1).
     */
    function length(UintSet storage set) internal view returns (uint256) {
        return _length(set._inner);
    }

   /**
    * @dev Returns the value stored at position `index` in the set. O(1).
    *
    * Note that there are no guarantees on the ordering of values inside the
    * array, and it may change when more values are added or removed.
    *
    * Requirements:
    *
    * - `index` must be strictly less than {length}.
    */
    function at(UintSet storage set, uint256 index) internal view returns (uint256) {
        return uint256(_at(set._inner, index));
    }
}
```

```
// File: localhost/contracts/libraries/MathV1.sol

pragma solidity >=0.5.0;

// a library for performing various math operations

library Math {
    function min(uint x, uint y) internal pure returns (uint z) {
        z = x < y ? x : y;
    }

    // babylonian method (https://en.wikipedia.org/wiki/Methods_of_computing_square_roots#Babylonian_method)
    function sqrt(uint y) internal pure returns (uint z) {
        if (y > 3) {
            z = y;
            uint x = y / 2 + 1;
            while (x < z) {
                z = x;
                x = (y / x + x) / 2;
            }
        } else if (y != 0) {
            z = 1;
        }
    }
}

// a library for performing overflow-safe math, courtesy of DappHub (https://github.com/dapphub/ds-math)
library SafeMath {
    function add(uint x, uint y) internal pure returns (uint z) {
        require((z = x + y) >= x, 'ds-math-add-overflow');
    }
```

```
    function sub(uint x, uint y) internal pure returns (uint z) {
        require((z = x - y) <= x, 'ds-math-sub-underflow');
    }

    function mul(uint x, uint y) internal pure returns (uint z) {
        require(y == 0 || (z = x * y) / y == x, 'ds-math-mul-overflow');
    }
}
// File: localhost/contracts/interfaces/IERC20.sol

//SPDX-License-Identifier: SimPL-2.0
pragma solidity >=0.5.0;

interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);

    /**
     * @dev Returns the amount of tokens owned by `account`.
     */
    function balanceOf(address account) external view returns (uint256);

    /**
     * @dev Moves `amount` tokens from the caller's account to `recipient`.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transfer(address recipient, uint256 amount) external returns (bool);
```

```
/**
 * @dev Returns the remaining number of tokens that `spender` will be
 * allowed to spend on behalf of `owner` through {transferFrom}. This is
 * zero by default.
 *
 * This value changes when {approve} or {transferFrom} are called.
 */
function allowance(address owner, address spender) external view returns (uint256);

/**
 * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
 *
 * Returns a boolean value indicating whether the operation succeeded.
 *
 * IMPORTANT: Beware that changing an allowance with this method brings the risk
 * that someone may use both the old and the new allowance by unfortunate
 * transaction ordering. One possible solution to mitigate this race
 * condition is to first reduce the spender's allowance to 0 and set the
 * desired value afterwards:
 * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
 *
 * Emits an {Approval} event.
 */
function approve(address spender, uint256 amount) external returns (bool);

/**
 * @dev Moves `amount` tokens from `sender` to `recipient` using the
 * allowance mechanism. `amount` is then deducted from the caller's
 * allowance.
 *
 * Returns a boolean value indicating whether the operation succeeded.
 *
```

```
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

    /**
     * @dev Emitted when the allowance of a `spender` for an `owner` is set by
     * a call to {approve}. `value` is the new allowance.
     */
    event Approval(address indexed owner, address indexed spender, uint256 value);
}
// File: localhost/contracts/token/Repurchase.sol

pragma solidity =0.5.16;

contract Ownable {

    address private _owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor () internal {
        _owner = msg.sender;
```

```
        emit OwnershipTransferred(address(0), msg.sender);
    }

    /**
     * @dev Returns the address of the current owner.
     */
    function owner() public view returns (address) {
        return _owner;
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(_owner == msg.sender, "YouSwap: CALLER_IS_NOT_THE_OWNER");
        _;
    }

    /**
     * @dev Leaves the contract without owner. It will not be possible to call
     * `onlyOwner` functions anymore. Can only be called by the current owner.
     *
     * NOTE: Renouncing ownership will leave the contract without an owner,
     * thereby removing any functionality that is only available to the owner.
     */
    function renounceOwnership() public    onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
```

```
     */
    function transferOwnership(address newOwner) public  onlyOwner {
        require(newOwner != address(0), "YouSwap:
NEW_OWNER_IS_THE_ZERO_ADDRESS");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
}



contract Repurchase is Ownable{
    using SafeMath for uint256;

    using EnumerableSet for EnumerableSet.AddressSet;
    EnumerableSet.AddressSet private _caller;

    bytes4 private constant SELECTOR = bytes4(keccak256(bytes('transfer(address,uint256)')));
    address public constant USDT = 0xFA8B1212119197eC88Fc768AF1b04aD0519Ad994;
    address public constant YOU = 0x1093EcdBACa4168F136f3BEfd17a261bfbbeDdA3;
    address public constant YOU_USDT = 0x94339BbdB9550a5758Da5EC65C547520EC819520;
    address public constant destroyAddress = 0xF971ec570538874F33cbc35C5156e9C3bC8CeF66;
    address public emergencyAddress;
    uint256 public amountIn;

    constructor (uint256 _amount, address _emergencyAddress) public {
        require(_amount > 0, "Amount must be greater than zero");
        require(_emergencyAddress != address(0), "Is zero address");
        amountIn = _amount;
        emergencyAddress = _emergencyAddress;
    }

    function setAmountIn(uint256 _newIn) public onlyOwner {
        amountIn = _newIn;
```

```
    }

    function setEmergencyAddress(address _newAddress) public onlyOwner {
        require(_newAddress != address(0), "Is zero address");
        emergencyAddress = _newAddress;
    }

    function addCaller(address _newCaller) public onlyOwner returns (bool) {
        require(_newCaller != address(0), "NewCaller is the zero address");
        return EnumerableSet.add(_caller, _newCaller);
    }

    function delCaller(address _delCaller) public onlyOwner returns (bool) {
        require(_delCaller != address(0), "DelCaller is the zero address");
        return EnumerableSet.remove(_caller, _delCaller);
    }

    function getCallerLength() public view returns (uint256) {
        return EnumerableSet.length(_caller);
    }

    function isCaller(address _call) public view returns (bool) {
        return EnumerableSet.contains(_caller, _call);
    }

    function getCaller(uint256 _index) public view returns (address){
        require(_index <= getCallerLength() - 1, "index out of bounds");
        return EnumerableSet.at(_caller, _index);
    }

    function swap() external onlyCaller returns (uint256 amountOut){
        require(IERC20(USDT).balanceOf(address(this)) >= amountIn, "Insufficient contract
balance");
```

```
        (uint256 reserve0, uint256 reserve1,) = IYouSwapPair(YOU_USDT).getReserves();
        uint256 amountInWithFee = amountIn.mul(997);
        amountOut = amountIn.mul(997).mul(reserve0) / reserve1.mul(1000).add(amountInWithFee);
        _safeTransfer(USDT, YOU_USDT, amountIn);
        IYouSwapPair(YOU_USDT).swap(amountOut, 0, destroyAddress, new bytes(0));
    }

    modifier onlyCaller() {
        require(isCaller(msg.sender), "Not the caller");
        _;
    }

    function emergencyWithdraw(address _token) public onlyOwner {
        require(IERC20(_token).balanceOf(address(this)) > 0, "Insufficient contract balance");
        IERC20(_token).transfer(emergencyAddress, IERC20(_token).balanceOf(address(this)));
    }

    function _safeTransfer(address token, address to, uint value) private {
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(SELECTOR, to,
value));
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'YouSwap:
TRANSFER_FAILED');
    }
}
```

IERC20.sol

```
//SPDX-License-Identifier: SimPL-2.0
pragma solidity >=0.5.0;

interface IERC20 {
    /**
     * @dev Returns the amount of tokens in existence.
     */
    function totalSupply() external view returns (uint256);
```

```
/**
 * @dev Returns the amount of tokens owned by `account`.
 */
function balanceOf(address account) external view returns (uint256);

/**
 * @dev Moves `amount` tokens from the caller's account to `recipient`.
 *
 * Returns a boolean value indicating whether the operation succeeded.
 *
 * Emits a {Transfer} event.
 */
function transfer(address recipient, uint256 amount) external returns (bool);

/**
 * @dev Returns the remaining number of tokens that `spender` will be
 * allowed to spend on behalf of `owner` through {transferFrom}. This is
 * zero by default.
 *
 * This value changes when {approve} or {transferFrom} are called.
 */
function allowance(address owner, address spender) external view returns (uint256);

/**
 * @dev Sets `amount` as the allowance of `spender` over the caller's tokens.
 *
 * Returns a boolean value indicating whether the operation succeeded.
 *
 * IMPORTANT: Beware that changing an allowance with this method brings the risk
 * that someone may use both the old and the new allowance by unfortunate
 * transaction ordering. One possible solution to mitigate this race
 * condition is to first reduce the spender's allowance to 0 and set the
```

```
     * desired value afterwards:
     * https://github.com/ethereum/EIPs/issues/20#issuecomment-263524729
     *
     * Emits an {Approval} event.
     */
    function approve(address spender, uint256 amount) external returns (bool);

    /**
     * @dev Moves `amount` tokens from `sender` to `recipient` using the
     * allowance mechanism. `amount` is then deducted from the caller's
     * allowance.
     *
     * Returns a boolean value indicating whether the operation succeeded.
     *
     * Emits a {Transfer} event.
     */
    function transferFrom(address sender, address recipient, uint256 amount) external returns (bool);

    /**
     * @dev Emitted when `value` tokens are moved from one account (`from`) to
     * another (`to`).
     *
     * Note that `value` may be zero.
     */
    event Transfer(address indexed from, address indexed to, uint256 value);

    /**
     * @dev Emitted when the allowance of a `spender` for an `owner` is set by
     * a call to {approve}. `value` is the new allowance.
     */
    event Approval(address indexed owner, address indexed spender, uint256 value);
}
```

IYouSwapCallee.sol

```
pragma solidity >=0.5.0;

interface IYouSwapCallee {
    function YouSwapV2Call(address sender, uint amount0, uint amount1, bytes calldata data) external;
}
```

IYouSwapERC20

```
pragma solidity >=0.5.0;

interface IYouSwapERC20 {
    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);

    function name() external pure returns (string memory);
    function symbol() external pure returns (string memory);
    function decimals() external pure returns (uint8);
    function totalSupply() external view returns (uint);
    function balanceOf(address owner) external view returns (uint);
    function allowance(address owner, address spender) external view returns (uint);

    function approve(address spender, uint value) external returns (bool);
    function transfer(address to, uint value) external returns (bool);
    function transferFrom(address from, address to, uint value) external returns (bool);

    function DOMAIN_SEPARATOR() external view returns (bytes32);
    function PERMIT_TYPEHASH() external pure returns (bytes32);
    function nonces(address owner) external view returns (uint);

    function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, bytes32
s) external;
}
```

IYouSwapFactory.sol

```
pragma solidity >=0.5.0;

interface IYouSwapFactory {
    event PairCreated(address indexed token0, address indexed token1, address pair, uint);

    function feeTo() external view returns (address);
    function feeToSetter() external view returns (address);

    function getPair(address tokenA, address tokenB) external view returns (address pair);
    function allPairs(uint) external view returns (address pair);
    function allPairsLength() external view returns (uint);

    function createPair(address tokenA, address tokenB) external returns (address pair);

    function setFeeTo(address) external;
    function setFeeToSetter(address) external;

    function setFeeToRate(uint256) external;
    function feeToRate() external view returns (uint256);
}
```

IYouSwapPair.sol

```
pragma solidity >=0.5.0;

interface IYouSwapPair {
    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);

    function name() external pure returns (string memory);
    function symbol() external pure returns (string memory);
    function decimals() external pure returns (uint8);
    function totalSupply() external view returns (uint);
    function balanceOf(address owner) external view returns (uint);
```

```
    function allowance(address owner, address spender) external view returns (uint);

    function approve(address spender, uint value) external returns (bool);
    function transfer(address to, uint value) external returns (bool);
    function transferFrom(address from, address to, uint value) external returns (bool);

    function DOMAIN_SEPARATOR() external view returns (bytes32);
    function PERMIT_TYPEHASH() external pure returns (bytes32);
    function nonces(address owner) external view returns (uint);

    function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, bytes32 s) external;

    event Mint(address indexed sender, uint amount0, uint amount1);
    event Burn(address indexed sender, uint amount0, uint amount1, address indexed to);
    event Swap(
        address indexed sender,
        uint amount0In,
        uint amount1In,
        uint amount0Out,
        uint amount1Out,
        address indexed to
    );
    event Sync(uint112 reserve0, uint112 reserve1);

    function MINIMUM_LIQUIDITY() external pure returns (uint);
    function factory() external view returns (address);
    function token0() external view returns (address);
    function token1() external view returns (address);
    function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32 blockTimestampLast);
    function price0CumulativeLast() external view returns (uint);
    function price1CumulativeLast() external view returns (uint);
```

```
    function kLast() external view returns (uint);

    function mint(address to) external returns (uint liquidity);
    function burn(address to) external returns (uint amount0, uint amount1);
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;
    function skim(address to) external;
    function sync() external;

    function initialize(address, address) external;
}
```

MathV1.sol

```
pragma solidity >=0.5.0;

// a library for performing various math operations

library Math {
    function min(uint x, uint y) internal pure returns (uint z) {
        z = x < y ? x : y;
    }

    // babylonian method (https://en.wikipedia.org/wiki/Methods_of_computing_square_roots#Babylonian_method)
    function sqrt(uint y) internal pure returns (uint z) {
        if (y > 3) {
            z = y;
            uint x = y / 2 + 1;
            while (x < z) {
                z = x;
                x = (y / x + x) / 2;
            }
        } else if (y != 0) {
            z = 1;
```

```
        }
    }
}

// a library for performing overflow-safe math, courtesy of DappHub (https://github.com/dapphub/ds-math)
library SafeMath {
    function add(uint x, uint y) internal pure returns (uint z) {
        require((z = x + y) >= x, 'ds-math-add-overflow');
    }

    function sub(uint x, uint y) internal pure returns (uint z) {
        require((z = x - y) <= x, 'ds-math-sub-underflow');
    }

    function mul(uint x, uint y) internal pure returns (uint z) {
        require(y == 0 || (z = x * y) / y == x, 'ds-math-mul-overflow');
    }
}
```

UQ112x112.sol

```
pragma solidity =0.5.16;

// a library for handling binary fixed point numbers (https://en.wikipedia.org/wiki/Q_(number_format))

// range: [0, 2**112 - 1]
// resolution: 1 / 2**112

library UQ112x112 {
    uint224 constant Q112 = 2**112;

    // encode a uint112 as a UQ112x112
    function encode(uint112 y) internal pure returns (uint224 z) {
        z = uint224(y) * Q112; // never overflows
```

```
    }

    // divide a UQ112x112 by a uint112, returning a UQ112x112
    function uqdiv(uint224 x, uint112 y) internal pure returns (uint224 z) {
        z = x / uint224(y);
    }
}
```

YouSwapERC20.sol

```
pragma solidity =0.5.16;

import './interfaces/IYouSwapERC20.sol';
import './libraries/MathV1.sol';

contract YouSwapERC20 is IYouSwapERC20 {
    using SafeMath for uint;

    string public constant name = 'YouSwap LP Token';
    string public constant symbol = 'ULP';
    uint8 public constant decimals = 18;
    uint  public totalSupply;
    mapping(address => uint) public balanceOf;
    mapping(address => mapping(address => uint)) public allowance;

    bytes32 public DOMAIN_SEPARATOR;
    // keccak256("Permit(address owner,address spender,uint256 value,uint256 nonce,uint256 deadline)");
    bytes32 public constant PERMIT_TYPEHASH = 0x6e71edae12b1b97f4d1f60370fef10105fa2faae0126114a169c64845d6126c9;
    mapping(address => uint) public nonces;

    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);
```

```
    constructor() public {
        uint chainId;
        assembly {
            chainId := chainid
        }
        DOMAIN_SEPARATOR = keccak256(
            abi.encode(
                keccak256('EIP712Domain(string name,string version,uint256 chainId,address verifyingContract)'),
                keccak256(bytes(name)),
                keccak256(bytes('1')),
                chainId,
                address(this)
            )
        );
    }

    function _mint(address to, uint value) internal {
        totalSupply = totalSupply.add(value);
        balanceOf[to] = balanceOf[to].add(value);
        emit Transfer(address(0), to, value);
    }

    function _burn(address from, uint value) internal {
        balanceOf[from] = balanceOf[from].sub(value);
        totalSupply = totalSupply.sub(value);
        emit Transfer(from, address(0), value);
    }

    function _approve(address owner, address spender, uint value) private {
        allowance[owner][spender] = value;
        emit Approval(owner, spender, value);
    }
```

```
    function _transfer(address from, address to, uint value) private {
        balanceOf[from] = balanceOf[from].sub(value);
        balanceOf[to] = balanceOf[to].add(value);
        emit Transfer(from, to, value);
    }

    function approve(address spender, uint value) external returns (bool) {
        _approve(msg.sender, spender, value);
        return true;
    }

    function transfer(address to, uint value) external returns (bool) {
        _transfer(msg.sender, to, value);
        return true;
    }

    function transferFrom(address from, address to, uint value) external returns (bool) {
        if (allowance[from][msg.sender] != uint(-1)) {
            allowance[from][msg.sender] = allowance[from][msg.sender].sub(value);
        }
        _transfer(from, to, value);
        return true;
    }

    function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, bytes32
s) external {
        require(deadline >= block.timestamp, 'YouSwap: EXPIRED');
        bytes32 digest = keccak256(
            abi.encodePacked(
                '\x19\x01',
                DOMAIN_SEPARATOR,
                keccak256(abi.encode(PERMIT_TYPEHASH, owner, spender, value,
```

```
nonces[owner]++, deadline))
                    )
            );
            address recoveredAddress = ecrecover(digest, v, r, s);
            require(recoveredAddress != address(0) && recoveredAddress == owner, 'YouSwap:
INVALID_SIGNATURE');
            _approve(owner, spender, value);
        }
}
```

YouSwapFactory,sol

```
pragma solidity =0.5.16;

import './interfaces/IYouSwapFactory.sol';
import './YouSwapPair.sol';
import './libraries/MathV1.sol';

contract YouSwapFactory is IYouSwapFactory {
    using SafeMath  for uint;

    address public feeTo;
    address public feeToSetter;
    uint256 public feeToRate;
    bytes32 public initCodeHash;

    mapping(address => mapping(address => address)) public getPair;
    address[] public allPairs;

    event PairCreated(address indexed token0, address indexed token1, address pair, uint);

    constructor(address _feeToSetter) public {
        feeToSetter = _feeToSetter;
        initCodeHash = keccak256(abi.encodePacked(type(YouSwapPair).creationCode));
```

```
    }

    function allPairsLength() external view returns (uint) {
        return allPairs.length;
    }

    function createPair(address tokenA, address tokenB) external returns (address pair) {
        require(tokenA != tokenB, 'YouSwap: IDENTICAL_ADDRESSES');
        (address token0, address token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
        require(token0 != address(0), 'YouSwap: ZERO_ADDRESS');
        require(getPair[token0][token1] == address(0), 'YouSwap: PAIR_EXISTS'); // single check is
sufficient
        bytes memory bytecode = type(YouSwapPair).creationCode;
        bytes32 salt = keccak256(abi.encodePacked(token0, token1));
        assembly {
            pair := create2(0, add(bytecode, 32), mload(bytecode), salt)
        }
        IYouSwapPair(pair).initialize(token0, token1);
        getPair[token0][token1] = pair;
        getPair[token1][token0] = pair; // populate mapping in the reverse direction
        allPairs.push(pair);
        emit PairCreated(token0, token1, pair, allPairs.length);
    }

    function setFeeTo(address _feeTo) external {
        require(msg.sender == feeToSetter, 'YouSwap: FORBIDDEN');
        feeTo = _feeTo;
    }

    function setFeeToSetter(address _feeToSetter) external {
        require(msg.sender == feeToSetter, 'YouSwap: FORBIDDEN');
        feeToSetter = _feeToSetter;
    }
```

```
    function setFeeToRate(uint256 _rate) external {
        require(msg.sender == feeToSetter, 'YouSwap: FORBIDDEN');
        require(_rate > 0, "YouSwap: FEE_TO_RATE_OVERFLOW");
        feeToRate = _rate.sub(1);
    }
}
```

YouSwapPair.sol

```
pragma solidity =0.5.16;

import './interfaces/IYouSwapPair.sol';
import './YouSwapERC20.sol';
import './libraries/MathV1.sol';
import './libraries/UQ112x112.sol';
import './interfaces/IERC20.sol';
import './interfaces/IYouSwapFactory.sol';
import './interfaces/IYouSwapCallee.sol';

contract YouSwapPair is IYouSwapPair, YouSwapERC20 {
    using SafeMath  for uint;
    using UQ112x112 for uint224;

    uint public constant MINIMUM_LIQUIDITY = 10**3;
    bytes4 private constant SELECTOR = bytes4(keccak256(bytes('transfer(address,uint256)')));

    address public factory;
    address public token0;
    address public token1;

    uint112 private reserve0;           // uses single storage slot, accessible via getReserves
    uint112 private reserve1;           // uses single storage slot, accessible via getReserves
    uint32  private blockTimestampLast; // uses single storage slot, accessible via getReserves
```

```
    uint public price0CumulativeLast;
    uint public price1CumulativeLast;
    uint public kLast; // reserve0 * reserve1, as of immediately after the most recent liquidity event

    uint private unlocked = 1;
    modifier lock() {
        require(unlocked == 1, 'YouSwap: LOCKED');
        unlocked = 0;
        _;
        unlocked = 1;
    }

    function getReserves() public view returns (uint112 _reserve0, uint112 _reserve1, uint32 _blockTimestampLast) {
        _reserve0 = reserve0;
        _reserve1 = reserve1;
        _blockTimestampLast = blockTimestampLast;
    }

    function _safeTransfer(address token, address to, uint value) private {
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(SELECTOR, to, value));
        require(success && (data.length == 0 || abi.decode(data, (bool))), 'YouSwap: TRANSFER_FAILED');
    }

    event Mint(address indexed sender, uint amount0, uint amount1);
    event Burn(address indexed sender, uint amount0, uint amount1, address indexed to);
    event Swap(
        address indexed sender,
        uint amount0In,
        uint amount1In,
        uint amount0Out,
```

```
        uint amount1Out,
        address indexed to
    );
    event Sync(uint112 reserve0, uint112 reserve1);

    constructor() public {
        factory = msg.sender;
    }

    // called once by the factory at time of deployment
    function initialize(address _token0, address _token1) external {
        require(msg.sender == factory, 'YouSwap: FORBIDDEN'); // sufficient check
        token0 = _token0;
        token1 = _token1;
    }

    // update reserves and, on the first call per block, price accumulators
    function _update(uint balance0, uint balance1, uint112 _reserve0, uint112 _reserve1) private {
        require(balance0 <= uint112(-1) && balance1 <= uint112(-1), 'YouSwap: OVERFLOW');
        uint32 blockTimestamp = uint32(block.timestamp % 2**32);
        uint32 timeElapsed = blockTimestamp - blockTimestampLast; // overflow is desired
        if (timeElapsed > 0 && _reserve0 != 0 && _reserve1 != 0) {
            // * never overflows, and + overflow is desired
            price0CumulativeLast += uint(UQ112x112.encode(_reserve1).uqdiv(_reserve0)) *
timeElapsed;
            price1CumulativeLast += uint(UQ112x112.encode(_reserve0).uqdiv(_reserve1)) *
timeElapsed;
        }
        reserve0 = uint112(balance0);
        reserve1 = uint112(balance1);
        blockTimestampLast = blockTimestamp;
        emit Sync(reserve0, reserve1);
    }
```

```
    // if fee is on, mint liquidity equivalent to 1/6th of the growth in sqrt(k)
    function _mintFee(uint112 _reserve0, uint112 _reserve1) private returns (bool feeOn) {
        address feeTo = IYouSwapFactory(factory).feeTo();
        feeOn = feeTo != address(0);
        uint _kLast = kLast; // gas savings
        if (feeOn) {
            if (_kLast != 0) {
                uint rootK = Math.sqrt(uint(_reserve0).mul(_reserve1));
                uint rootKLast = Math.sqrt(_kLast);
                if (rootK > rootKLast) {
                    uint numerator = totalSupply.mul(rootK.sub(rootKLast));
                    uint denominator =
rootK.mul(IYouSwapFactory(factory).feeToRate()).add(rootKLast);
                    uint liquidity = numerator / denominator;
                    if (liquidity > 0) _mint(feeTo, liquidity);
                }
            }
        } else if (_kLast != 0) {
            kLast = 0;
        }
    }

    // this low-level function should be called from a contract which performs important safety checks
    function mint(address to) external lock returns (uint liquidity) {
        (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
        uint balance0 = IERC20(token0).balanceOf(address(this));
        uint balance1 = IERC20(token1).balanceOf(address(this));
        uint amount0 = balance0.sub(_reserve0);
        uint amount1 = balance1.sub(_reserve1);

        bool feeOn = _mintFee(_reserve0, _reserve1);
        uint _totalSupply = totalSupply; // gas savings, must be defined here since totalSupply can
```

```
update in _mintFee
        if (_totalSupply == 0) {
            liquidity = Math.sqrt(amount0.mul(amount1)).sub(MINIMUM_LIQUIDITY);
           _mint(address(0), MINIMUM_LIQUIDITY); // permanently lock the first MINIMUM_LIQUIDITY tokens
        } else {
            liquidity = Math.min(amount0.mul(_totalSupply) / _reserve0, amount1.mul(_totalSupply) / _reserve1);
        }
        require(liquidity > 0, 'YouSwap: INSUFFICIENT_LIQUIDITY_MINTED');
        _mint(to, liquidity);

        _update(balance0, balance1, _reserve0, _reserve1);
        if (feeOn) kLast = uint(reserve0).mul(reserve1); // reserve0 and reserve1 are up-to-date
        emit Mint(msg.sender, amount0, amount1);
    }

    // this low-level function should be called from a contract which performs important safety checks
    function burn(address to) external lock returns (uint amount0, uint amount1) {
        (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
        address _token0 = token0;                                // gas savings
        address _token1 = token1;                                // gas savings
        uint balance0 = IERC20(_token0).balanceOf(address(this));
        uint balance1 = IERC20(_token1).balanceOf(address(this));
        uint liquidity = balanceOf[address(this)];

        bool feeOn = _mintFee(_reserve0, _reserve1);
        uint _totalSupply = totalSupply; // gas savings, must be defined here since totalSupply can update in _mintFee
        amount0 = liquidity.mul(balance0) / _totalSupply; // using balances ensures pro-rata distribution
        amount1 = liquidity.mul(balance1) / _totalSupply; // using balances ensures pro-rata distribution
```

```
        require(amount0 > 0 && amount1 > 0, 'YouSwap: INSUFFICIENT_LIQUIDITY_BURNED');
        _burn(address(this), liquidity);
        _safeTransfer(_token0, to, amount0);
        _safeTransfer(_token1, to, amount1);
        balance0 = IERC20(_token0).balanceOf(address(this));
        balance1 = IERC20(_token1).balanceOf(address(this));

        _update(balance0, balance1, _reserve0, _reserve1);
        if (feeOn) kLast = uint(reserve0).mul(reserve1); // reserve0 and reserve1 are up-to-date
        emit Burn(msg.sender, amount0, amount1, to);
    }

    // this low-level function should be called from a contract which performs important safety checks
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external lock {
        require(amount0Out > 0 || amount1Out > 0, 'YouSwap: INSUFFICIENT_OUTPUT_AMOUNT');
        (uint112 _reserve0, uint112 _reserve1,) = getReserves(); // gas savings
        require(amount0Out < _reserve0 && amount1Out < _reserve1, 'YouSwap: INSUFFICIENT_LIQUIDITY');

        uint balance0;
        uint balance1;
        { // scope for _token{0,1}, avoids stack too deep errors
        address _token0 = token0;
        address _token1 = token1;
        require(to != _token0 && to != _token1, 'YouSwap: INVALID_TO');
        if (amount0Out > 0) _safeTransfer(_token0, to, amount0Out); // optimistically transfer tokens
        if (amount1Out > 0) _safeTransfer(_token1, to, amount1Out); // optimistically transfer tokens
        if (data.length > 0) IYouSwapCallee(to).YouSwapV2Call(msg.sender, amount0Out, amount1Out, data);
        balance0 = IERC20(_token0).balanceOf(address(this));
        balance1 = IERC20(_token1).balanceOf(address(this));
```

```
        }
        uint amount0In = balance0 > _reserve0 - amount0Out ? balance0 - (_reserve0 - amount0Out) :
0;
        uint amount1In = balance1 > _reserve1 - amount1Out ? balance1 - (_reserve1 - amount1Out) :
0;
        require(amount0In > 0 || amount1In > 0, 'YouSwap: INSUFFICIENT_INPUT_AMOUNT');
        { // scope for reserve{0,1}Adjusted, avoids stack too deep errors
        uint balance0Adjusted = balance0.mul(1000).sub(amount0In.mul(3));
        uint balance1Adjusted = balance1.mul(1000).sub(amount1In.mul(3));
        require(balance0Adjusted.mul(balance1Adjusted)                                    >=
uint(_reserve0).mul(_reserve1).mul(1000**2), 'YouSwap: K');
        }

        _update(balance0, balance1, _reserve0, _reserve1);
        emit Swap(msg.sender, amount0In, amount1In, amount0Out, amount1Out, to);
    }

    // force balances to match reserves
    function skim(address to) external lock {
        address _token0 = token0; // gas savings
        address _token1 = token1; // gas savings
        _safeTransfer(_token0, to, IERC20(_token0).balanceOf(address(this)).sub(reserve0));
        _safeTransfer(_token1, to, IERC20(_token1).balanceOf(address(this)).sub(reserve1));
    }

    // force reserves to match balances
    function sync() external lock {
        _update(IERC20(token0).balanceOf(address(this)),
IERC20(token1).balanceOf(address(this)), reserve0, reserve1);
    }
}
```

YouSwapRouter.sol

```
/**
 *Submitted for verification at Etherscan.io on 2021-03-13
*/

// File: localhost/contracts/interfaces/ISwapMining.sol

pragma solidity >=0.5.0;

interface ISwapMining {
    function swap(address account, address input, address output, uint256 amount) external returns (bool);
}

// File: localhost/contracts/interfaces/IWETH.sol

pragma solidity >=0.5.0;

interface IWETH {
    function deposit() external payable;
    function transfer(address to, uint value) external returns (bool);
    function withdraw(uint) external;
}

// File: localhost/contracts/interfaces/IERC20.sol

pragma solidity >=0.5.0;

interface IERC20 {
    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);

    function name() external view returns (string memory);
    function symbol() external view returns (string memory);
```

```
    function decimals() external view returns (uint8);
    function totalSupply() external view returns (uint);
    function balanceOf(address owner) external view returns (uint);
    function allowance(address owner, address spender) external view returns (uint);

    function approve(address spender, uint value) external returns (bool);
    function transfer(address to, uint value) external returns (bool);
    function transferFrom(address from, address to, uint value) external returns (bool);
}

// File: localhost/contracts/libraries/SafeMath.sol

pragma solidity =0.6.6;

// a library for performing overflow-safe math, courtesy of DappHub (https://github.com/dapphub/ds-math)

library SafeMath {
    function add(uint x, uint y) internal pure returns (uint z) {
        require((z = x + y) >= x, 'ds-math-add-overflow');
    }

    function sub(uint x, uint y) internal pure returns (uint z) {
        require((z = x - y) <= x, 'ds-math-sub-underflow');
    }

    function mul(uint x, uint y) internal pure returns (uint z) {
        require(y == 0 || (z = x * y) / y == x, 'ds-math-mul-overflow');
    }
}

// File: localhost/contracts/interfaces/IYouSwapPair.sol
```

```
pragma solidity >=0.5.0;

interface IYouSwapPair {
    event Approval(address indexed owner, address indexed spender, uint value);
    event Transfer(address indexed from, address indexed to, uint value);

    function name() external pure returns (string memory);
    function symbol() external pure returns (string memory);
    function decimals() external pure returns (uint8);
    function totalSupply() external view returns (uint);
    function balanceOf(address owner) external view returns (uint);
    function allowance(address owner, address spender) external view returns (uint);

    function approve(address spender, uint value) external returns (bool);
    function transfer(address to, uint value) external returns (bool);
    function transferFrom(address from, address to, uint value) external returns (bool);

    function DOMAIN_SEPARATOR() external view returns (bytes32);
    function PERMIT_TYPEHASH() external pure returns (bytes32);
    function nonces(address owner) external view returns (uint);

    function permit(address owner, address spender, uint value, uint deadline, uint8 v, bytes32 r, bytes32
s) external;

    event Mint(address indexed sender, uint amount0, uint amount1);
    event Burn(address indexed sender, uint amount0, uint amount1, address indexed to);
    event Swap(
        address indexed sender,
        uint amount0In,
        uint amount1In,
        uint amount0Out,
        uint amount1Out,
        address indexed to
```

```
    );
    event Sync(uint112 reserve0, uint112 reserve1);

    function MINIMUM_LIQUIDITY() external pure returns (uint);
    function factory() external view returns (address);
    function token0() external view returns (address);
    function token1() external view returns (address);
    function getReserves() external view returns (uint112 reserve0, uint112 reserve1, uint32 blockTimestampLast);
    function price0CumulativeLast() external view returns (uint);
    function price1CumulativeLast() external view returns (uint);
    function kLast() external view returns (uint);

    function mint(address to) external returns (uint liquidity);
    function burn(address to) external returns (uint amount0, uint amount1);
    function swap(uint amount0Out, uint amount1Out, address to, bytes calldata data) external;
    function skim(address to) external;
    function sync() external;

    function initialize(address, address) external;
}

// File: localhost/contracts/libraries/YouSwapLibrary.sol

pragma solidity >=0.5.0;



library YouSwapLibrary {
    using SafeMath for uint;

    // returns sorted token addresses, used to handle return values from pairs sorted in this order
    function sortTokens(address tokenA, address tokenB) internal pure returns (address token0, address
```

```
token1) {
        require(tokenA != tokenB, 'YouSwapLibrary: IDENTICAL_ADDRESSES');
        (token0, token1) = tokenA < tokenB ? (tokenA, tokenB) : (tokenB, tokenA);
        require(token0 != address(0), 'YouSwapLibrary: ZERO_ADDRESS');
    }

    // calculates the CREATE2 address for a pair without making any external calls
    function pairFor(address factory, address tokenA, address tokenB) internal pure returns (address
pair) {
        (address token0, address token1) = sortTokens(tokenA, tokenB);
        pair = address(uint(keccak256(abi.encodePacked(
                hex'ff',
                factory,
                keccak256(abi.encodePacked(token0, token1)),
                hex'8919347964f406fcc7e9a98fd3e05e8ba3e0270039e1e056121a8bffd0f2789e'   //
init code hash
            ))));
    }

    // fetches and sorts the reserves for a pair
    function getReserves(address factory, address tokenA, address tokenB) internal view returns (uint
reserveA, uint reserveB) {
        (address token0,) = sortTokens(tokenA, tokenB);
        (uint reserve0, uint reserve1,) = IYouSwapPair(pairFor(factory, tokenA,
tokenB)).getReserves();
        (reserveA, reserveB) = tokenA == token0 ? (reserve0, reserve1) : (reserve1, reserve0);
    }

    // given some amount of an asset and pair reserves, returns an equivalent amount of the other asset
    function quote(uint amountA, uint reserveA, uint reserveB) internal pure returns (uint amountB) {
        require(amountA > 0, 'YouSwapLibrary: INSUFFICIENT_AMOUNT');
        require(reserveA > 0 && reserveB > 0, 'YouSwapLibrary: INSUFFICIENT_LIQUIDITY');
        amountB = amountA.mul(reserveB) / reserveA;
```

```
    }

    // given an input amount of an asset and pair reserves, returns the maximum output amount of the other asset
    function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut) internal pure returns (uint amountOut) {
        require(amountIn > 0, 'YouSwapLibrary: INSUFFICIENT_INPUT_AMOUNT');
        require(reserveIn > 0 && reserveOut > 0, 'YouSwapLibrary: INSUFFICIENT_LIQUIDITY');
        uint amountInWithFee = amountIn.mul(997);
        uint numerator = amountInWithFee.mul(reserveOut);
        uint denominator = reserveIn.mul(1000).add(amountInWithFee);
        amountOut = numerator / denominator;
    }

    // given an output amount of an asset and pair reserves, returns a required input amount of the other asset
    function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut) internal pure returns (uint amountIn) {
        require(amountOut > 0, 'YouSwapLibrary: INSUFFICIENT_OUTPUT_AMOUNT');
        require(reserveIn > 0 && reserveOut > 0, 'YouSwapLibrary: INSUFFICIENT_LIQUIDITY');
        uint numerator = reserveIn.mul(amountOut).mul(1000);
        uint denominator = reserveOut.sub(amountOut).mul(997);
        amountIn = (numerator / denominator).add(1);
    }

    // performs chained getAmountOut calculations on any number of pairs
    function getAmountsOut(address factory, uint amountIn, address[] memory path) internal view returns (uint[] memory amounts) {
        require(path.length >= 2, 'YouSwapLibrary: INVALID_PATH');
        amounts = new uint[](path.length);
        amounts[0] = amountIn;
        for (uint i; i < path.length - 1; i++) {
            (uint reserveIn, uint reserveOut) = getReserves(factory, path[i], path[i + 1]);
```

```
            amounts[i + 1] = getAmountOut(amounts[i], reserveIn, reserveOut);
        }
    }

    // performs chained getAmountIn calculations on any number of pairs
    function getAmountsIn(address factory, uint amountOut, address[] memory path) internal view returns (uint[] memory amounts) {
        require(path.length >= 2, 'YouSwapLibrary: INVALID_PATH');
        amounts = new uint[](path.length);
        amounts[amounts.length - 1] = amountOut;
        for (uint i = path.length - 1; i > 0; i--) {
            (uint reserveIn, uint reserveOut) = getReserves(factory, path[i - 1], path[i]);
            amounts[i - 1] = getAmountIn(amounts[i], reserveIn, reserveOut);
        }
    }
}

// File: localhost/contracts/interfaces/IYouSwapRouter.sol

pragma solidity >=0.6.2;

interface IYouSwapRouter {
    function factory() external pure returns (address);
    function WETH() external pure returns (address);
    function swapMining() external pure returns (address);

    function addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
```

```
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB, uint liquidity);
    function addLiquidityETH(
        address token,
        uint amountTokenDesired,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external payable returns (uint amountToken, uint amountETH, uint liquidity);
    function removeLiquidity(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external returns (uint amountA, uint amountB);
    function removeLiquidityETH(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external returns (uint amountToken, uint amountETH);
    function removeLiquidityWithPermit(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
```

```
        uint amountBMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountA, uint amountB);
    function removeLiquidityETHWithPermit(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountToken, uint amountETH);
    function swapExactTokensForTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external returns (uint[] memory amounts);
    function swapTokensForExactTokens(
        uint amountOut,
        uint amountInMax,
        address[] calldata path,
        address to,
        uint deadline
    ) external returns (uint[] memory amounts);
    function swapExactETHForTokens(uint amountOutMin, address[] calldata path, address to, uint
deadline)
        external
        payable
        returns (uint[] memory amounts);
```

function swapTokensForExactETH(uint amountOut, uint amountInMax, address[] calldata path, address to, uint deadline)
external
returns (uint[] memory amounts);
function swapExactTokensForETH(uint amountIn, uint amountOutMin, address[] calldata path, address to, uint deadline)
external
returns (uint[] memory amounts);
function swapETHForExactTokens(uint amountOut, address[] calldata path, address to, uint deadline)
external
payable
returns (uint[] memory amounts);

function quote(uint amountA, uint reserveA, uint reserveB) external pure returns (uint amountB);
function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut) external pure returns (uint amountOut);
function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut) external pure returns (uint amountIn);
function getAmountsOut(uint amountIn, address[] calldata path) external view returns (uint[] memory amounts);
function getAmountsIn(uint amountOut, address[] calldata path) external view returns (uint[] memory amounts);

function removeLiquidityETHSupportingFeeOnTransferTokens(
address token,
uint liquidity,
uint amountTokenMin,
uint amountETHMin,
address to,
uint deadline
) external returns (uint amountETH);
function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(

```
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external returns (uint amountETH);

    function swapExactTokensForTokensSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external;
    function swapExactETHForTokensSupportingFeeOnTransferTokens(
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external payable;
    function swapExactTokensForETHSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    ) external;
}

// File: localhost/contracts/libraries/Ownable.sol
```

```
//SPDX-License-Identifier: SimPL-2.0
pragma solidity ^0.6.0;

/**
 * @dev Contract module which provides a basic access control mechanism, where
 * there is an account (an owner) that can be granted exclusive access to
 * specific functions.
 *
 * By default, the owner account will be the one that deploys the contract. This
 * can later be changed with {transferOwnership}.
 *
 * This module is used through inheritance. It will make available the modifier
 * `onlyOwner`, which can be applied to your functions to restrict their use to
 * the owner.
 */
contract Ownable {

    address private _owner;

    event OwnershipTransferred(address indexed previousOwner, address indexed newOwner);

    /**
     * @dev Initializes the contract setting the deployer as the initial owner.
     */
    constructor () internal {
        _owner = msg.sender;
        emit OwnershipTransferred(address(0), msg.sender);
    }

    /**
     * @dev Returns the address of the current owner.
     */
    function owner() public view returns (address) {
```

```
        return _owner;
    }

    /**
     * @dev Throws if called by any account other than the owner.
     */
    modifier onlyOwner() {
        require(_owner == msg.sender, "YouSwap: CALLER_IS_NOT_THE_OWNER");
        _;
    }

    /**
     * @dev Leaves the contract without owner. It will not be possible to call
     * `onlyOwner` functions anymore. Can only be called by the current owner.
     *
     * NOTE: Renouncing ownership will leave the contract without an owner,
     * thereby removing any functionality that is only available to the owner.
     */
    function renounceOwnership() public virtual onlyOwner {
        emit OwnershipTransferred(_owner, address(0));
        _owner = address(0);
    }

    /**
     * @dev Transfers ownership of the contract to a new account (`newOwner`).
     * Can only be called by the current owner.
     */
    function transferOwnership(address newOwner) public virtual onlyOwner {
        require(newOwner != address(0), "YouSwap:
NEW_OWNER_IS_THE_ZERO_ADDRESS");
        emit OwnershipTransferred(_owner, newOwner);
        _owner = newOwner;
    }
```

```
}
// File: localhost/contracts/libraries/TransferHelper.sol

// SPDX-License-Identifier: GPL-3.0-or-later

pragma solidity >=0.6.0;

// helper methods for interacting with ERC20 tokens and sending ETH that do not consistently return true/false
library TransferHelper {
    function safeApprove(
        address token,
        address to,
        uint256 value
    ) internal {
        // bytes4(keccak256(bytes('approve(address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x095ea7b3, to, value));
        require(
            success && (data.length == 0 || abi.decode(data, (bool))),
            'TransferHelper::safeApprove: approve failed'
        );
    }

    function safeTransfer(
        address token,
        address to,
        uint256 value
    ) internal {
        // bytes4(keccak256(bytes('transfer(address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0xa9059cbb, to, value));
        require(
```

```
            success && (data.length == 0 || abi.decode(data, (bool))),
            'TransferHelper::safeTransfer: transfer failed'
        );
    }

    function safeTransferFrom(
        address token,
        address from,
        address to,
        uint256 value
    ) internal {
        // bytes4(keccak256(bytes('transferFrom(address,address,uint256)')));
        (bool success, bytes memory data) = token.call(abi.encodeWithSelector(0x23b872dd, from, to,
value));
        require(
            success && (data.length == 0 || abi.decode(data, (bool))),
            'TransferHelper::transferFrom: transferFrom failed'
        );
    }

    function safeTransferETH(address to, uint256 value) internal {
        (bool success, ) = to.call{value: value}(new bytes(0));
        require(success, 'TransferHelper::safeTransferETH: ETH transfer failed');
    }
}

// File: localhost/contracts/interfaces/IYouSwapFactory.sol

pragma solidity >=0.5.0;

interface IYouSwapFactory {
    event PairCreated(address indexed token0, address indexed token1, address pair, uint);
```

```
    function feeTo() external view returns (address);
    function feeToSetter() external view returns (address);

    function getPair(address tokenA, address tokenB) external view returns (address pair);
    function allPairs(uint) external view returns (address pair);
    function allPairsLength() external view returns (uint);

    function createPair(address tokenA, address tokenB) external returns (address pair);

    function setFeeTo(address) external;
    function setFeeToSetter(address) external;

    function setFeeToRate(uint256) external;
    function feeToRate() external view returns (uint256);
}

// File: localhost/contracts/YouSwapRouter.sol

pragma solidity =0.6.6;



contract YouSwapRouter is IYouSwapRouter, Ownable {
    using SafeMath for uint;

    address public immutable override factory;
```

```
address public immutable override WETH;
address public override swapMining;

modifier ensure(uint deadline) {
    require(deadline >= block.timestamp, 'YouSwapRouter: EXPIRED');
    _;
}

constructor(address _factory, address _WETH) public {
    factory = _factory;
    WETH = _WETH;
}

receive() external payable {
    assert(msg.sender == WETH); // only accept ETH via fallback from the WETH contract
}

function setSwapMining(address _swapMininng) public onlyOwner {
    swapMining = _swapMininng;
}

// **** ADD LIQUIDITY ****
function _addLiquidity(
    address tokenA,
    address tokenB,
    uint amountADesired,
    uint amountBDesired,
    uint amountAMin,
    uint amountBMin
) internal virtual returns (uint amountA, uint amountB) {
    // create the pair if it doesn't exist yet
    if (IYouSwapFactory(factory).getPair(tokenA, tokenB) == address(0)) {
        IYouSwapFactory(factory).createPair(tokenA, tokenB);
```

```
        }
        (uint reserveA, uint reserveB) = YouSwapLibrary.getReserves(factory, tokenA, tokenB);
        if (reserveA == 0 && reserveB == 0) {
            (amountA, amountB) = (amountADesired, amountBDesired);
        } else {
            uint amountBOptimal = YouSwapLibrary.quote(amountADesired, reserveA, reserveB);
            if (amountBOptimal <= amountBDesired) {
                require(amountBOptimal >= amountBMin, 'YouSwapRouter: INSUFFICIENT_B_AMOUNT');
                (amountA, amountB) = (amountADesired, amountBOptimal);
            } else {
                uint amountAOptimal = YouSwapLibrary.quote(amountBDesired, reserveB, reserveA);
                assert(amountAOptimal <= amountADesired);
                require(amountAOptimal >= amountAMin, 'YouSwapRouter: INSUFFICIENT_A_AMOUNT');
                (amountA, amountB) = (amountAOptimal, amountBDesired);
            }
        }
    }
    function addLiquidity(
        address tokenA,
        address tokenB,
        uint amountADesired,
        uint amountBDesired,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) external virtual override ensure(deadline) returns (uint amountA, uint amountB, uint liquidity) {
        (amountA, amountB) = _addLiquidity(tokenA, tokenB, amountADesired, amountBDesired, amountAMin, amountBMin);
        address pair = YouSwapLibrary.pairFor(factory, tokenA, tokenB);
```

```
        TransferHelper.safeTransferFrom(tokenA, msg.sender, pair, amountA);
        TransferHelper.safeTransferFrom(tokenB, msg.sender, pair, amountB);
        liquidity = IYouSwapPair(pair).mint(to);
    }
    function addLiquidityETH(
        address token,
        uint amountTokenDesired,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) external virtual override payable ensure(deadline) returns (uint amountToken, uint amountETH,
uint liquidity) {
        (amountToken, amountETH) = _addLiquidity(
            token,
            WETH,
            amountTokenDesired,
            msg.value,
            amountTokenMin,
            amountETHMin
        );
        address pair = YouSwapLibrary.pairFor(factory, token, WETH);
        TransferHelper.safeTransferFrom(token, msg.sender, pair, amountToken);
        IWETH(WETH).deposit{value: amountETH}();
        assert(IWETH(WETH).transfer(pair, amountETH));
        liquidity = IYouSwapPair(pair).mint(to);
        // refund dust eth, if any
        if (msg.value > amountETH) TransferHelper.safeTransferETH(msg.sender, msg.value -
amountETH);
    }

    // **** REMOVE LIQUIDITY ****
    function removeLiquidity(
```

```
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline
    ) public virtual override ensure(deadline) returns (uint amountA, uint amountB) {
        address pair = YouSwapLibrary.pairFor(factory, tokenA, tokenB);
        IYouSwapPair(pair).transferFrom(msg.sender, pair, liquidity); // send liquidity to pair
        (uint amount0, uint amount1) = IYouSwapPair(pair).burn(to);
        (address token0,) = YouSwapLibrary.sortTokens(tokenA, tokenB);
        (amountA, amountB) = tokenA == token0 ? (amount0, amount1) : (amount1, amount0);
        require(amountA >= amountAMin, 'YouSwapRouter: INSUFFICIENT_A_AMOUNT');
        require(amountB >= amountBMin, 'YouSwapRouter: INSUFFICIENT_B_AMOUNT');
    }
    function removeLiquidityETH(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) public virtual override ensure(deadline) returns (uint amountToken, uint amountETH) {
        (amountToken, amountETH) = removeLiquidity(
            token,
            WETH,
            liquidity,
            amountTokenMin,
            amountETHMin,
            address(this),
            deadline
        );
```

```
        TransferHelper.safeTransfer(token, to, amountToken);
        IWETH(WETH).withdraw(amountETH);
        TransferHelper.safeTransferETH(to, amountETH);
    }
    function removeLiquidityWithPermit(
        address tokenA,
        address tokenB,
        uint liquidity,
        uint amountAMin,
        uint amountBMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external virtual override returns (uint amountA, uint amountB) {
        address pair = YouSwapLibrary.pairFor(factory, tokenA, tokenB);
        uint value = approveMax ? uint(-1) : liquidity;
        IYouSwapPair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
        (amountA, amountB) = removeLiquidity(tokenA, tokenB, liquidity, amountAMin,
amountBMin, to, deadline);
    }
    function removeLiquidityETHWithPermit(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
    ) external virtual override returns (uint amountToken, uint amountETH) {
        address pair = YouSwapLibrary.pairFor(factory, token, WETH);
        uint value = approveMax ? uint(-1) : liquidity;
        IYouSwapPair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
        (amountToken, amountETH) = removeLiquidityETH(token, liquidity, amountTokenMin,
```

```
amountETHMin, to, deadline);
    }

    // **** REMOVE LIQUIDITY (supporting fee-on-transfer tokens) ****
    function removeLiquidityETHSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline
    ) public virtual override ensure(deadline) returns (uint amountETH) {
        (, amountETH) = removeLiquidity(
            token,
            WETH,
            liquidity,
            amountTokenMin,
            amountETHMin,
            address(this),
            deadline
        );
        TransferHelper.safeTransfer(token, to, IERC20(token).balanceOf(address(this)));
        IWETH(WETH).withdraw(amountETH);
        TransferHelper.safeTransferETH(to, amountETH);
    }
    function removeLiquidityETHWithPermitSupportingFeeOnTransferTokens(
        address token,
        uint liquidity,
        uint amountTokenMin,
        uint amountETHMin,
        address to,
        uint deadline,
        bool approveMax, uint8 v, bytes32 r, bytes32 s
```

```
    ) external virtual override returns (uint amountETH) {
        address pair = YouSwapLibrary.pairFor(factory, token, WETH);
        uint value = approveMax ? uint(-1) : liquidity;
        IYouSwapPair(pair).permit(msg.sender, address(this), value, deadline, v, r, s);
        amountETH = removeLiquidityETHSupportingFeeOnTransferTokens(
            token, liquidity, amountTokenMin, amountETHMin, to, deadline
        );
    }

    // **** SWAP ****
    // requires the initial amount to have already been sent to the first pair
    function _swap(uint[] memory amounts, address[] memory path, address _to) internal virtual {
        for (uint i; i < path.length - 1; i++) {
            (address input, address output) = (path[i], path[i + 1]);
            (address token0,) = YouSwapLibrary.sortTokens(input, output);
            uint amountOut = amounts[i + 1];
            if (swapMining != address(0)) {
                ISwapMining(swapMining).swap(msg.sender, input, output, amountOut);
            }
            (uint amount0Out, uint amount1Out) = input == token0 ? (uint(0), amountOut) :
(amountOut, uint(0));
            address to = i < path.length - 2 ? YouSwapLibrary.pairFor(factory, output, path[i + 2]) :
_to;
            IYouSwapPair(YouSwapLibrary.pairFor(factory, input, output)).swap(
                amount0Out, amount1Out, to, new bytes(0)
            );
        }
    }
    function swapExactTokensForTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
```

```
        uint deadline
    ) external virtual override ensure(deadline) returns (uint[] memory amounts) {
        amounts = YouSwapLibrary.getAmountsOut(factory, amountIn, path);
        require(amounts[amounts.length - 1] >= amountOutMin, 'YouSwapRouter: INSUFFICIENT_OUTPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, YouSwapLibrary.pairFor(factory, path[0], path[1]), amounts[0]
        );
        _swap(amounts, path, to);
    }
    function swapTokensForExactTokens(
        uint amountOut,
        uint amountInMax,
        address[] calldata path,
        address to,
        uint deadline
    ) external virtual override ensure(deadline) returns (uint[] memory amounts) {
        amounts = YouSwapLibrary.getAmountsIn(factory, amountOut, path);
        require(amounts[0] <= amountInMax, 'YouSwapRouter: EXCESSIVE_INPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, YouSwapLibrary.pairFor(factory, path[0], path[1]), amounts[0]
        );
        _swap(amounts, path, to);
    }
    function swapExactETHForTokens(uint amountOutMin, address[] calldata path, address to, uint deadline)
        external
        virtual
        override
        payable
        ensure(deadline)
        returns (uint[] memory amounts)
    {
```

```
        require(path[0] == WETH, 'YouSwapRouter: INVALID_PATH');
        amounts = YouSwapLibrary.getAmountsOut(factory, msg.value, path);
        require(amounts[amounts.length - 1] >= amountOutMin, 'YouSwapRouter: INSUFFICIENT_OUTPUT_AMOUNT');
        IWETH(WETH).deposit{value: amounts[0]}();
        assert(IWETH(WETH).transfer(YouSwapLibrary.pairFor(factory, path[0], path[1]), amounts[0]));
        _swap(amounts, path, to);
    }
    function swapTokensForExactETH(uint amountOut, uint amountInMax, address[] calldata path, address to, uint deadline)
        external
        virtual
        override
        ensure(deadline)
        returns (uint[] memory amounts)
    {
        require(path[path.length - 1] == WETH, 'YouSwapRouter: INVALID_PATH');
        amounts = YouSwapLibrary.getAmountsIn(factory, amountOut, path);
        require(amounts[0] <= amountInMax, 'YouSwapRouter: EXCESSIVE_INPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, YouSwapLibrary.pairFor(factory, path[0], path[1]), amounts[0]
        );
        _swap(amounts, path, address(this));
        IWETH(WETH).withdraw(amounts[amounts.length - 1]);
        TransferHelper.safeTransferETH(to, amounts[amounts.length - 1]);
    }
    function swapExactTokensForETH(uint amountIn, uint amountOutMin, address[] calldata path, address to, uint deadline)
        external
        virtual
        override
        ensure(deadline)
```

```
        returns (uint[] memory amounts)
    {
        require(path[path.length - 1] == WETH, 'YouSwapRouter: INVALID_PATH');
        amounts = YouSwapLibrary.getAmountsOut(factory, amountIn, path);
        require(amounts[amounts.length - 1] >= amountOutMin, 'YouSwapRouter: INSUFFICIENT_OUTPUT_AMOUNT');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, YouSwapLibrary.pairFor(factory, path[0], path[1]), amounts[0]
        );
        _swap(amounts, path, address(this));
        IWETH(WETH).withdraw(amounts[amounts.length - 1]);
        TransferHelper.safeTransferETH(to, amounts[amounts.length - 1]);
    }
    function swapETHForExactTokens(uint amountOut, address[] calldata path, address to, uint deadline)
        external
        virtual
        override
        payable
        ensure(deadline)
        returns (uint[] memory amounts)
    {
        require(path[0] == WETH, 'YouSwapRouter: INVALID_PATH');
        amounts = YouSwapLibrary.getAmountsIn(factory, amountOut, path);
        require(amounts[0] <= msg.value, 'YouSwapRouter: EXCESSIVE_INPUT_AMOUNT');
        IWETH(WETH).deposit{value: amounts[0]}();
        assert(IWETH(WETH).transfer(YouSwapLibrary.pairFor(factory, path[0], path[1]), amounts[0]));
        _swap(amounts, path, to);
        // refund dust eth, if any
        if (msg.value > amounts[0]) TransferHelper.safeTransferETH(msg.sender, msg.value - amounts[0]);
    }
```

```
    // **** SWAP (supporting fee-on-transfer tokens) ****
    // requires the initial amount to have already been sent to the first pair
    function _swapSupportingFeeOnTransferTokens(address[] memory path, address _to) internal virtual {
        for (uint i; i < path.length - 1; i++) {
            (address input, address output) = (path[i], path[i + 1]);
            (address token0,) = YouSwapLibrary.sortTokens(input, output);
            IYouSwapPair pair = IYouSwapPair(YouSwapLibrary.pairFor(factory, input, output));
            uint amountInput;
            uint amountOutput;
            { // scope to avoid stack too deep errors
            (uint reserve0, uint reserve1,) = pair.getReserves();
            (uint reserveInput, uint reserveOutput) = input == token0 ? (reserve0, reserve1) : (reserve1, reserve0);
            amountInput = IERC20(input).balanceOf(address(pair)).sub(reserveInput);
            amountOutput = YouSwapLibrary.getAmountOut(amountInput, reserveInput, reserveOutput);
            }
            if (swapMining != address(0)) {
                ISwapMining(swapMining).swap(msg.sender, input, output, amountOutput);
            }
            (uint amount0Out, uint amount1Out) = input == token0 ? (uint(0), amountOutput) : (amountOutput, uint(0));
            address to = i < path.length - 2 ? YouSwapLibrary.pairFor(factory, output, path[i + 2]) : _to;
            pair.swap(amount0Out, amount1Out, to, new bytes(0));
        }
    }
    function swapExactTokensForTokensSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
```

```
        address to,
        uint deadline
    ) external virtual override ensure(deadline) {
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, YouSwapLibrary.pairFor(factory, path[0], path[1]), amountIn
        );
        uint balanceBefore = IERC20(path[path.length - 1]).balanceOf(to);
        _swapSupportingFeeOnTransferTokens(path, to);
        require(
            IERC20(path[path.length - 1]).balanceOf(to).sub(balanceBefore) >= amountOutMin,
            'YouSwapRouter: INSUFFICIENT_OUTPUT_AMOUNT'
        );
    }
    function swapExactETHForTokensSupportingFeeOnTransferTokens(
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    )
        external
        virtual
        override
        payable
        ensure(deadline)
    {
        require(path[0] == WETH, 'YouSwapRouter: INVALID_PATH');
        uint amountIn = msg.value;
        IWETH(WETH).deposit{value: amountIn}();
        assert(IWETH(WETH).transfer(YouSwapLibrary.pairFor(factory, path[0], path[1]),
amountIn));
        uint balanceBefore = IERC20(path[path.length - 1]).balanceOf(to);
        _swapSupportingFeeOnTransferTokens(path, to);
        require(
```

```
            IERC20(path[path.length - 1]).balanceOf(to).sub(balanceBefore) >= amountOutMin,
            'YouSwapRouter: INSUFFICIENT_OUTPUT_AMOUNT'
        );
    }
    function swapExactTokensForETHSupportingFeeOnTransferTokens(
        uint amountIn,
        uint amountOutMin,
        address[] calldata path,
        address to,
        uint deadline
    )
        external
        virtual
        override
        ensure(deadline)
    {
        require(path[path.length - 1] == WETH, 'YouSwapRouter: INVALID_PATH');
        TransferHelper.safeTransferFrom(
            path[0], msg.sender, YouSwapLibrary.pairFor(factory, path[0], path[1]), amountIn
        );
        _swapSupportingFeeOnTransferTokens(path, address(this));
        uint amountOut = IERC20(WETH).balanceOf(address(this));
        require(amountOut >= amountOutMin, 'YouSwapRouter: INSUFFICIENT_OUTPUT_AMOUNT');
        IWETH(WETH).withdraw(amountOut);
        TransferHelper.safeTransferETH(to, amountOut);
    }

    // **** LIBRARY FUNCTIONS ****
    function quote(uint amountA, uint reserveA, uint reserveB) public pure virtual override returns (uint amountB) {
        return YouSwapLibrary.quote(amountA, reserveA, reserveB);
    }
```

```
function getAmountOut(uint amountIn, uint reserveIn, uint reserveOut)
        public
        pure
        virtual
        override
        returns (uint amountOut)
{
        return YouSwapLibrary.getAmountOut(amountIn, reserveIn, reserveOut);
}

function getAmountIn(uint amountOut, uint reserveIn, uint reserveOut)
        public
        pure
        virtual
        override
        returns (uint amountIn)
{
        return YouSwapLibrary.getAmountIn(amountOut, reserveIn, reserveOut);
}

function getAmountsOut(uint amountIn, address[] memory path)
        public
        view
        virtual
        override
        returns (uint[] memory amounts)
{
        return YouSwapLibrary.getAmountsOut(factory, amountIn, path);
}

function getAmountsIn(uint amountOut, address[] memory path)
        public
```

```
        view
        virtual
        override
        returns (uint[] memory amounts)
    {
        return YouSwapLibrary.getAmountsIn(factory, amountOut, path);
    }
}
```

6. 附录 B：安全风险评级标准

智能合约漏洞评级标准	
漏洞评级	**漏洞评级说明**
高危漏洞	能直接造成代币合约或用户资金损失的漏洞，如：能造成代币价值归零的数值溢出漏洞、能造成交易所损失代币的假充值漏洞、能造成合约账户损失 TRX 或代币的重入漏洞等； 能造成代币合约归属权丢失的漏洞，如：关键函数的访问控制缺陷、call 注入导致关键函数访问控制绕过等； 能造成代币合约无法正常工作的漏洞，如：因向恶意地址发送 TRX 导致的拒绝服务漏洞、因 energy 耗尽导致的拒绝服务漏洞。
中危漏洞	需要特定地址才能触发的高风险漏洞，如代币合约拥有者才能触发的数值溢出漏洞等；非关键函数的访问控制缺陷、不能造成直接资金损失的逻辑设计缺陷等。
低危漏洞	难以被触发的漏洞、触发之后危害有限的漏洞，如需要大量 TRX 或代币才能触发的数值溢出漏洞、触发数值溢出后攻击者无法直接获利的漏洞、通过指定高 energy 触发的事务顺序依赖风险等。

7. 附录 C：智能合约安全审计工具简介

7.1 Manticore

Manticore 是一个分析二进制文件和智能合约的符号执行工具，Manticore 包含一个符号虚拟机（EVM），一个 EVM 反汇编器/汇编器以及一个用于自动编译和分析 Solidity 的方便界面。它还集成了 Ethersplay，用于 EVM 字节码的 Bit of Traits of Bits 可视化反汇编程序，用于可视化分析。与二进制文件一样，Manticore 提供了一个简单的命令行界面和一个用于分析 EVM 字节码的 Python API。

7.2 Oyente

Oyente 是一个智能合约分析工具，Oyente 可以用来检测智能合约中常见的 bug，比如 reentrancy、事务排序依赖等等。更方便的是，Oyente 的设计是模块化的，所以这让高级用户可以实现并插入他们自己的检测逻辑，以检查他们的合约中自定义的属性。

7.3 securify.sh

Securify 可以验证智能合约常见的安全问题，例如交易乱序和缺少输入验证，它在全自动化的同时分析程序所有可能的执行路径，此外，Securify 还具有用于指定漏洞的特定语言，这使 Securify 能够随时关注当前的安全性和其他可靠性问题。

7.4 Echidna

Echidna 是一个为了对 EVM 代码进行模糊测试而设计的 Haskell 库。

7.5 MAIAN

MAIAN 是一个用于查找智能合约漏洞的自动化工具，Maian 处理合约的字

节码，并尝试建立一系列交易以找出并确认错误。

7.6 ethersplay

ethersplay 是一个 EVM 反汇编器，其中包含了相关分析工具。

7.7 ida-evm

ida-evm 是一个针对虚拟机（EVM）的 IDA 处理器模块。

7.8 Remix-ide

Remix 是一款基于浏览器的编译器和 IDE，可让用户使用 Solidity 语言构建合约并调试交易。

7.9 知道创宇区块链安全审计人员专用工具包

知道创宇安全审计人员专用工具包，由知道创宇渗透测试工程师研发，收集和使用，包含专用于测试人员的批量自动测试工具，自主研发的工具、脚本或利用工具等。

知道创宇

北京知道创宇信息技术股份有限公司

咨询电话 +86(10)400 060 9587

邮　　箱 sec@knownsec.com

官　　网 www.knownsec.com

地　　址 北京市 朝阳区 望京 SOHO T2-B座-2509